cgdb-manual-in-chinese

leeyiw

Published
with GitBook

Table of Contents

1. Introduction
2. CGDB简介
3. 1 打开和关闭CGDB
4. 2 理解CGDB的几个核心概念
    i. 2.1 理解代码窗口
    ii. 2.2 理解GDB窗口
    iii. 2.3 理解文件对话框窗口
    iv. 2.4 理解TTY窗口
    v. 2.5 理解状态栏
    vi. 2.6 在不同的窗口中切换
5. 3 CGDB命令
    i. 3.1 CGDB模式中的命令
    ii. 3.2 GDB模式中的命令
    iii. 3.3 文件对话框模式中的命令
    iv. 3.4 TTY模式中的命令
6. 4 CGDB配置命令
7. 5 CGDB高亮组
    i. 5.1 不同的高亮组
    ii. 5.2 不同的属性
    iii. 5.3 不同的颜色
8. 6 CGDB键盘用户接口
    i. 6.1 KUI超时选项
    ii. 6.2 使用映射
    iii. 6.3 键码
9. 7 对被调试的程序进行I/O操作

关于《CGDB中文手册》

《CGDB中文手册》是一本由英文版《CGDB Manual》翻译而来的一本手册，CGDB的项目主页在http://cgdb.github.com/，英文原版的《CGDB Manual》在http://cgdb.github.com/docs/cgdb.html。

翻译这本手册的想法是我在学习CGDB的时候产生的，CGDB是一款非常优秀的gdb调试器的前端，但是网上的资料匮乏，以至于只有一本英文手册比较详细。为了能够让更多人能够更方便、快速的学习这个工具，于是我产生了翻译的想法。

欢迎各位对CGDB有兴趣的朋友一起参加本手册的翻译，由于我们的英文水平不高，在翻译的过程中难免产生大量的错误，也请您对手册中翻译有误的地方提出宝贵意见。

授权许可

本书中的内容使用CC BY-SA 4.0 International License（知识共享署名-相同方式共享 4.0 国际许可协议）授权。

参与者

感谢下列译者为本书的翻译作出的贡献，他们是：

- 李毅为（翻译第1～5章）
- Peixu.Zhu（翻译第6.2、6.3节）
- 武飞（校对）
- diseng（校对）

在 GitBook 上阅读本书

开始阅读

上一章：目录，下一章：打开和关闭CGDB，目录：目录

CGDB简介

CGDB是一个基于curses图形库的GNU Debugger（GDB）图形接口。CGDB的目标是变的轻量而且敏捷，并且不会加入不必要的功能使其变得臃肿。

CGDB的图形接口是参考GDB的文本用户接口（tui）设计和实现的，它使用一个分屏显示了当前执行的代码。代码区的界面模仿了Unix经典的文本编辑器：vi。熟悉vi的人对CGDB应该有着宾至如归的感觉。

CGDB中负责和GDB通信的库是Trivial GDB（tgdb或者叫做libtgdb）。使用这个抽象层使得展示代码的UI界面能够独立于调试器，并且极大的简化了CGDB的实现。

推荐那些开发其他GDB的接口的人使用libtgdb作为程序的抽象层。使用它可以避免很多令人头疼的GDB的输出和注释的解析问题。

CGDB具有如下这些特性：

- 高亮的源代码窗口
- 可视化的断点
- 常用功能的键盘快捷键
- 搜索源代码（通过正则表达式）

上一章：CGDB简介，下一章：理解CGDB的几个核心概念，目录：目录

1 打开和关闭CGDB

本章讨论的是如何进入和退出CGDB。有如下几种方法：

- 在命令行下输入 *'cgdb'* 运行CGDB
- 在GDB窗口输入 *'quit'* 或者按下 *'Ctrl+D'* 退出CGDB
- 在源代码窗口输入 *':quit'* 也可以退出CGDB。这在GDB挂起或者运行一条耗时很长的指令时也同样起作用

上一章：打开和关闭CGDB，下一节：理解代码窗口，目录：目录

2 理解CGDB的几个核心概念

CGDB的界面默认由两个窗口和一条状态栏组成。代码窗口默认在上方，GDB窗口默认在下方。状态栏在两个窗口之间。

根据不同窗口的激活情况，CGDB有不同的模式。当代码窗口被激活时，CGDB处于CGDB模式。当GDB窗口被激活时，CGDB处于GDB模式。当TTY窗口被激活时，CGDB处于TTY模式。

当CGDB发布1.0的版本时，所有的窗口将会是可移动的，用户可以创建想要的窗口，个数不限。但是目前我的时间都花在了开发CGDB和GDB之间的接口上。当这部分完成的时候，CGDB的界面会变得更完美。如果你是一个熟悉ncurses的开发者，并且有业余时间来做这部分工作的话，请联系我。

上一节：理解CGDB的几个核心概念，下一节：理解GDB窗口，目录：目录

2.1 理解代码窗口

您可以通过代码窗口查看当前被调试程序的源代码。CGDB只能同时显示一个源文件。当用户调试程序的时候，通过 *next* 和 *step* 命令，CGDB将会更新源代码以及行号，以此提醒您调试进行到了何处。

CGDB有几个新特性能让调试比使用旧的GDB更方便。其中，当您在调试C，C++或ADA程序的时候，源代码是高亮的。这个特性可以让您更加快速的找到源文件中的特定代码。如果您需要让CGDB在调试其他语言程序的时候能够显示代码高亮，请联系我们。一些代码窗口的使用命令参见3.1节。

除了显示源代码，CGDB也显示当前运行的代码行。当前被GDB运行的代码行的行号被高亮成绿色，同时CGDB也在这一行的行号前显示一个箭头。您可以通过配置选项 *':set arrowstyle'* 来配置箭头的样式。默认的情况下，使用值 *short* 配置的短箭头生效。但是我个人更加喜欢使用值 *long* 配置的长箭头。

当您浏览代码窗口时，光标所在行的行号被高亮成了白色。它使得您可以知道光标处在当前文件的什么位置。

您可以通过CGDB的代码窗口设置或者取消一个断点。浏览至您需要设置断点的行，然后按下空格键，断点就会被设置。当断点成功被设置时，行号会被标红。再按下一次空格键断点就会被取消。当断点被禁用的时候，行号会被标黄。

CGDB还支持在代码窗口中使用正则表达式搜索源代码。您可以按下 / 或者 *?*，然后加上您要搜索的字符串或者是正则表达式。这里的搜索使用的是C语言的正则表达式库，使得在搜索 '\|' 或 '+'* 时表现的更快。

代码窗口中可使用的全部命令的列表在3.1节。

上一节：理解代码窗口，下一节：理解文件对话框窗口，目录：目录

2.2 理解GDB窗口

CGDB通过GDB窗口让用户直接操作gdb。如果您想要将一个命令发送给gdb，只要将命令输入进GDB窗口即可。使用CGDB中的GDB窗口和在命令行下使用gdb是完全一样的。

在这个窗口中输入的一部分快捷键会被CGDB截获并处理，而非直接发送给gdb。这些快捷键的列表在3.2节

类似于终端操作，CGDB将会将您连续输入的多个命令缓存起来。因此当您在一个命令完成之前又输入了多个其他的命令，这些命令会被CGDB按照顺序依次执行。就像在gdb中调试一样，除非按下 *Ctrl-C* ，否则缓存的命令不会被停止执行。

上一节：理解GDB窗口，下一节：理解TTY窗口，目录：目录

2.3 理解文件对话框窗口

通过文件对话框窗口，用户可以浏览和选择他们想要查看的文件。它为用户提供了一个与被调试程序有关的所有源代码文件的列表。如果没有文件被显示，可能是因为没有程序正在被调试，或者是因为被调试的可执行程序中没有调试信息，在这些情况下，文件对话框窗口不会打开，状态栏上会显示一条错误信息。

您可以通过在代码窗口中键入 *o* 来打开文件对话框窗口。当您打开了文件对话框窗口后，需要通过键入 *q* 来关闭它。您可以通过方向键来选择您想要的文件，甚至可以使用正则表达式去搜索文件。在文件数量越来越多的情况下，这个功能可以省去您很多的时间。

文件对话框窗口中可使用的全部命令的列表在3.3节。

上一节：理解文件对话框窗口，下一节：理解状态栏，目录：目录

2.4 理解TTY窗口

用户可以通过TTY窗口将程序的输入传给被调试的程序。这个窗口与GDB窗口类似，除了在TTY窗口中输入的数据将被直接传给被调试的程序。参见第七章。

您会看到，在TTY窗口和被调试的程序窗口之间有一个tty设备。因此如果被调试的程序使用readline等行缓冲输入，则命令行输入是可以被编辑的。这个tty设备也会被通过TTY窗口当作程序的终端输出。您可以在TTY窗口和GDB窗口之间看到这个tty设备的名字。

TTY窗口中可使用的全部命令的列表在3.4节。

对被调试的程序进行的I/O容易让人迷惑。这部分在第七章有更好的解释。除非被调试的程序的I/O比较简单，否则我通常都会选择在另外一个终端启动被调试程序，然后在CGDB中连接上被调试程序。

上一节：理解TTY窗口，下一节：在不同的窗口中切换，目录：目录

2.5 理解状态栏

状态栏是CGDB向用户显示正在输入的命令和当错误发生时报告错误的最通用的方式。CGDB不使用弹窗和其它形式的输出来告知用户信息或有错误发生。

当CGDB运行的时候，您可以通过任何在CGDB配置文件中有效的配置命令去配置CGDB。只需要代码窗口键入 *:* 您就可以看见一个冒号出现在状态栏中，此后输入的命令也会被显示在状态栏中。当您输入了您想要执行的命令后，键入 *enter* 。这将会让CGDB运行您之前键入的命令。如果您在输入命令的时候想要取消您输入的内容，可以按下CGDB的模式选择键。这样做会让您回到CGDB模式。更详细的关于CGDB的模式选择键参见2.6节。

状态栏中可使用的全部命令的列表在第4章。

上一节：理解状态栏，下一章：CGDB命令，目录：目录

2.6 在不同的窗口中切换

当CGDB最初运行时，CGDB默认处于GDB模式。状态栏右侧的 ** 符号显示当前的输入会被传入*GDB*窗口。按下 ESC 键可切换至代码窗口，*CGDB*模式切换键是将用户从别的模式切换到*CGDB*模式的快捷键，默认的*CGDB*模式切换键是 ESC* 。如果您想改变CGDB模式切换键对应的快捷键，请查阅CGDB的配置选项。参见第4章。

现在CGDB处于CGDB模式。要切换回GDB模式，键入 *i* 。这个设置是基于最流行的Unix文本编辑器：vi。

上一节：在不同的窗口中切换，下一节：CGDB模式中的命令，目录：目录

3 CGDB命令

您可以通过多种方式控制CGDB。每个模式下有不同的控制CGDB的方式。目前CGDB会根据哪一个窗口被激活，隐式的切换到对应模式。接下来您将会看到不同模式中可以使用的一些命令。

上一节：CGDB命令，下一节：GDB模式中的命令，目录：目录

3.1 CGDB模式中的命令

当您处于代码窗口时，您已经处于CGDB模式中。本节中所有的命令在这个模式中都起作用。这个模式主要是让用户可以浏览当前调试的源代码文件，进行搜索和切换到别的模式中。

`cgdbmodekey`
让用户进入命令模式。但是，处于CGDB模式中代表您已经在此模式。这个按键默认为 *ESC* 键

`i`
让用户进入GDB模式

`I`
让用户进入TTY模式

`T`
打开一个窗口并将输入传递给被调试的程序

`Ctrl+T`
为被调试程序打开一个新的tty

`k`
`up arrow`
向上移动一行

`j`
`down arrow`
向下移动一行

`h`
`left arrow`
向左移动一列

`l`
`right arrow`
向右移动一列

`Ctrl-b`
`page up`
向上翻一页

`Ctrl-u`
向上翻半页

`Ctrl-f`
`page down`
向下翻一页

`Ctrl-d`
向下翻半页

`gg`
移动到文件顶部

`G`
移动到文件底部

`/`

从当前光标处向下搜索

`?`
从当前光标处向上搜索

`n`
继续向下搜索

`N`
继续向上搜索

`o`
打开文件对话框窗口

`spacebar`
在当前行设置断点

`t`
在当前行设置一个临时断点

`-`
将代码窗口缩小一行

`=`
将代码窗口增大一行

`_`
将代码窗口缩小25%（当TTY窗口显示的时候，将TTY窗口缩小一行）

`+`
将代码窗口增大25%（当TTY窗口显示的时候，将TTY窗口增大一行）

`Ctrl-l`
清屏并重绘

`F5`
发送一个run命令至GDB

`F6`
发送一个continue命令至GDB

`F7`
发送一个finish命令至GDB

`F8`
发送一个next命令至GDB

`F10`
发送一个step命令至GDB

上一节：CGDB模式中的命令，下一节：文件对话框模式中的命令，目录：目录

3.2 GDB模式中的命令

在GDB模式中，用户最常关注的是GDB控制台的使用。也就是向GDB发送命令与接受GDB返回的消息。几乎所有传入这个窗口的数据都是直接发送给readline缓冲区，然后发送给GDB。

CGDB会首先对关心的按键输入进行处理，然后再将其余的发送给GDB，理解这点是非常重要的。CGDB会处理一些它关心的按键输入。以下是CGDB会首先处理的按键，而且这些按键输入并不会被继续传递给GDB。

`cgdbmodekey`
切换至代码窗口。这个按键默认为 *ESC* 键

`page up` 向上翻一页

`page down`
向下翻一页

`F11`
移动到GDB输出的头部

`F12`
移动到GDB输出的尾部

对于除了以上列出的其他输入，CGDB目前并不关心。CGDB将会直接将这些输入传递给readline库。当readline判断接收到一个命令时，它将通知CGDB并且将命令发送给GDB。这和直接使用GDB使用的处理方法是一样的。

上一节：GDB模式中的命令，下一节：TTY模式中的命令，目录：目录

3.3 文件对话框模式中的命令

文件对话框主要用来让用户查找和打开被调试程序的源代码文件。文件对话框是全屏的，且会将每个被调试程序的源代码文件都列出。一个常见的文件对话框会在源代码窗口按下 *o* 键时被打开，然后可以搜索相关的文件。比如，如果您在查找 *foo.c*，可以输入 */foo.c*，然后按下回车键结束正则表达式的输入，选择您需要的文件。

在文件对话框模式中可用的命令如下：

`q`
将会退出文件对话框并返回代码窗口

`k`
`up arrow`
向上移动一行

`j`
`down arrow`
向下移动一行

`h`
`left arrow`
向左移动一行

`l`
`right arrow`
向右移动一行

`Ctrl-b`
`page up`
向上翻一页

`Ctrl-f` `page down`
向下翻一页

`/`
从当前光标处向下搜索

`?`
从当前光标处向上搜索

`n`
继续向下搜索

`N`
继续向上搜索

`enter`
选择当前的文件

上一节：文件对话框模式中的命令，下一章：CGDB配置命令，目录：目录

3.4 TTY模式中的命令

`cgdbmodekey`
返回代码窗口。这个按键默认为 *ESC* 键

`page up`
向上翻一页

`page down`
向下翻一页

`F11`
移动到GDB输出的头部

`F12`
移动到GDB输出的尾部

上一节：TTY模式中的命令，下一章：CGDB高亮组，目录：目录

4 CGDB配置命令

您可能会觉得CGDB中的一些特性比较有用。CGDB可以通过一个叫做 *cgdbrc* 的文件自动加载CGDB的命令。CGDB将会在 *$HOME/.cgdb/* 目录下查找这个文件。如果这个文件存在的话，CGDB将会依次运行这个文件中的每一行。这和用户在tui初始化后在状态栏输入所有的命令是一样的。

下列变量改变了CGDB一些方面的行为。其中有些命令是缩写，所有的布尔命令都可能通过在命令前添加 *'no'* 而变成一个否定的命令。例如： `:set ignorecase` 将搜索设置为大小写不敏感；而设置 `:set noignorecase` 则将搜索设置为大小写敏感。

`:set as=` *style*
`:set arrowstyle=` *style*
设置箭头的风格。可能的取值为 *'short'*， *'long'*，以及 *'highlight'* 。被改变风格的箭头是指向当前运行的源代码行的箭头。默认值是 *'short'*。为了更容易阅读，CGDB提供了更长的箭头。最后， *'highlight'* 选项不绘制出箭头，而是将整行反色。

`:set asr`
`:set autosourcereload`
如果这个选项被打开了，CGDB将会在源代码文件被CGDB打开后又被修改的时候自动地重新加载源代码文件。如果这个选项被关闭了，源代码文件将不会被重新加载，直到您重启CGDB。这个选项默认是开启的。这个特性在您调试程序时，修改源代码，重新编译以及在GDB的命令行窗口中输入 *r* 时非常有用。在这种情况下，这个文件将会被更新到最新的版本。请注意，CGDB只通过检查源代码文件的时间戳来判断它是否被改变。因此如果您修改了代码，但是没有重新编译它，CGDB将会依然重新加载改动后的源代码。

`:set cgdbmodekey=` *key*
这个选项用来设置将CGDB切换至CGDB模式的快捷键。默认的情况下， *ESC* 键是CGDB模式键。通过使用键码，CGDB模式键还可以被设置为任意的其他键。这个选项在用户想使用readline的vi模式时会特别有用。如果用户输入 `:set cgdbmodekey=<PageUp>` 然后 *PageUp* 键将会让CGDB进入CGDB模式， 而 *ESC* 键将会被readline接收。

`:set ic`
`:set ignorecase`
将搜索设置为大小写不敏感。默认情况下，这个选项是关闭的（默认大小写敏感）。

`:set stc`
`:set showtgdbcommands`
当这个选项被开启时，CGDB将会显示所有它发送给GDB的命令。如果这个选项关闭，CGDB将不会显示它发送给GDB的命令。这个选项默认是关闭的。

`:set syn=` *style*
`:set syntax=` *style*
将当前的源代码设置为 *style* 类型的高亮模式。可取的值有 *'c'* ， *'ada'* 以及 *'off'* 。通常的情况下，用户不会使用到这个命令，因为CGDB会自动得通过检测源代码文件的扩展名来选择高亮模式。但是这个特性目前可以被用于调试CGDB。

`:set to`
`:set timeout`
这个选项是与 *ttimeout* 选项一起使用的，它用来决定CGDB在接收到一部分被映射的按键或者是一部分的虚拟键码后的行为。如果这个选项被开启了，CGDB将会在一段时间后将这些按键序列或是虚拟键码的一部分判断为超时。如果这个选项是关闭的，用户定义的映射将不会被判断为超时。CGDB将会通过检验 *ttimeout* 选项的值来决定是否将虚拟键码的部分输入判断为超时。想要确定CGDB会如何处理被映射的按键与虚拟键码的超时和超时时长，请参见第六章中的列表。这个选项默认是被开启的。

`:set tm=` *delay*
`:set timeoutlen=` *delay*
这个选项被用来与 *ttimeoutlen* 选项配合使用。它用毫秒数值表示了CGDB将会等待一个按键映射序列完成输入的时间长度。

如果 *delay* 为0，CGDB将会立即接收每个收到的字符输入。这样会阻止任何按键映射或是虚拟键码的输入。*delay* 可以在0至10000之间取值，包括0和10000。*delay* 的默认值为1000（一秒）。

`:set ttimeout`
这个选项是与 *timeout* 一起使用的，它用来决定CGDB在接收到一部分虚拟键码后的行为。如果这个选项被开启了，CGDB对从键盘输入的虚拟键码设置超时。如果这个选项关闭，则CGDB将会根据 *timeout* 选项来决定是否需要对虚拟键码设置超时。想要确定CGDB会如何处理虚拟键码的超时和超时时长，请参见第六章中的列表。这个选项默认是被开启的。

`:set ttm=` *delay*
`:set ttimeoutlen=` *delay*
这个选项是与 *timeoutlen* 选项一起使用的。它用毫秒数值表示了CGDB将会等待一个虚拟键码被完成输入的时间长度。如果 *delay* 为0，CGDB将会立即接收每个收到的字符输入。这样将会阻止任何虚拟键码的输入。*delay* 可以在0至10000之间取值，包括0和10000。*delay* 的默认值为100（十分之一秒）。

`:set ts=` *number*
`:set tabstop=` *number*
设置一个TAB键在屏幕上显示的对应空格的个数。默认的 *number* 值为8。

`:set wmh=` *number*
`:set winminheight=` *number*
窗口的最小高度。CGDB中的所有的窗口的高度都不会小于这个值。默认的 *number* 值为0。

`:set winsplit=` *style*
设置代码窗口和GDB窗口分界的位置。这个选项在被写入cgdbrc时会非常有用。参见第四章。*style* 选项可取的值有 *top_full*， *top_big*， *even*， *bottom_big* 以及 *bottom_full*。

`:set ws`
`:set wrapscan`
当搜索到文件的末尾时从头开始继续搜索。这个选项默认是被开启的。

`:c`
`:continue`
向GDB发送一个 `continue` 命令。

`:down`
向GDB发送一个 `down` 命令。

`:e`
`:edit`
重新加载代码窗口中的文件。这个选项在文件被cgdb打开后被改变的时候有用。

`:f`
`:finish`
向GDB发送一个 `finish` 命令。

`:help`
这个命令将在代码窗口中以文本形式显示本手册。

`:hi` *group* `cterm=` *attributes* `ctermfg=` *color* `ctermbg=` *color* `term=` *attributes*
`:highlight` *group* `cterm=` *attributes* `ctermfg=` *color* `ctermbg=` *color* `term=` *attributes*
为特定的高亮组设置颜色和属性。命令的格式模仿了vim中的“highlight”命令。 `group` 、 `attributes` 和 `color` 的可以选择的值请参见第五章。
您可以给出任意顺序、任意数量的名字-值对。'*ctermfg*' 和 '*ctermbg*' 分别设置前景色和背景色。可以通过数字或者vim中颜色的名字来选择特定的颜色。当CGDB与ncurses链接时，使用数字表达颜色的数值可以设置为-1至COLORS之间的值。当CGDB与curses链接时，数值必须在0至COLORS之间。
'*cterms*' 设置彩色终端的视频属性。'*term*' 设置黑白终端的视频属性。以下是一些这个命令的例子：
`:highlight Logo cterm=bold,underline ctermfg=Red ctermbg=Black`
`:highlight Normal cterm=reverse ctermfg=White ctermbg=Black`
`:hi Normal term=bold`

`:insert`
激活GDB窗口。

`:n`
`:next`
向GDB发送一个next命令。

`:q`
`:quit`
退出CGDB。

`:r`
`:run`
向GDB发送一个run命令。

`:start`
向GDB发送一个start命令。

`:k`
`:kill`
向GDB发送一个kill命令。

`:s`
`:step`
向GDB发送一个step命令。

`:syntax`
打开或关闭代码高亮（使用`:syntax on`与`:syntax off`打开和关闭代码高亮，译者注）

`:up`
向GDB发送一个up命令。

`:map` *lhs rhs*
创建一个在CGDB模式下的新的键盘映射，或是替换一个已有的CGDB模式下的键盘映射。在命令被执行后，当 *lhs* 被输入时，CGDB将会收到 *rhs* 而不是 *lhs*。更多关于如何使用map命令的细节请参见6.2节。

`:unm lhs`
`:unmap lhs`
删除一个CGDB模式下的已有的键盘映射。*lhs* 是用户在创建键盘映射时左边输入的字符组合。例如，如果用户输入`:map a<Space>b foo`，那么用户可以通过输入`:unmap a<Space>b`来取消这个已经存在的键盘映射。

`:im lhs rhs`
`:imap lhs rhs`
创建一个GDB模式下的新的键盘映射，或是替换一个已有的GDB模式下的键盘映射。在命令被执行后，当 *lhs* 被输入时，CGDB将会收到 *rhs* 而不是 *lhs*。更多关于如何使用map命令的细节请参见6.2节。

`:iu lhs`
`:iunmap lhs`
删除一个GDB模式下的已有的键盘映射。*lhs* 是用户在创建键盘映射时左边输入的字符组合。例如，如果用户输入`:imap a<Space>b foo`，那么用户可以通过输入`:iunmap a<Space>b`来取消这个已经存在的键盘映射。

上一章：CGDB配置命令，下一节：不同的高亮组，目录：目录

5 CGDB高亮组

如果终端支持彩色显示，那么CGDB是能够使用彩色的。直到0.6.1版本，CGDB不允许用户以任何方式配置色彩高亮。目前CGDB中的色彩高亮是完全可配置的。

CGDB在色彩高亮上模仿了vim中的色彩高亮。任何在终端中被高亮的数据都是被一个高亮组所表示的。一个高亮组表示了数据需要被高亮的前景色、背景色和属性。目前CGDB中有多种不同类型的高亮组。有语法高亮组，代表着对源代码的语法高亮。也有UI高亮组，代表CGDB的logo与状态栏等等。

每个高亮组都有一组默认的属性和颜色与其相关联。您可以通过highlight命令来修改一个高亮组的属性。参见第四章。

请注意，CGDB目前支持使用和启动CGDB前终端的颜色相同的背景颜色。但是，这只在CGDB与ncurses链接时有效。如果您将CGDB与curses链接，则CGDB将会将背景色强制设置成黑色。

- 可使用的高亮组：不同的高亮组。
- 可使用的属性：不同的属性。
- 可使用的颜色：不同的颜色。

上一节：CGDB高亮组，下一节：不同的属性，目录：目录

5.1 不同的高亮组

下面的列表列出了CGDB在高亮代码文件时会使用的全部高亮组。

`Statement`
表示了一种语言定义的关键字

`Type`
表示了一种语言定义的类型

`Constant`
表示了字符串或者数字

`Comment`
表示了源代码中的注释

`PreProc`
表示了C/C++中的预处理指令

`Normal`
表示了普通文本

下面的列表列出了CGDB在显示其UI时会使用的全部高亮组。

`StatusLine`
表示了CGDB中的状态栏。文件对话框的状态栏也会是用这个高亮组。

`IncSearch`
表示了当用户在文件对话框窗口或是代码窗口中搜索时使用的高亮组。

`Arrow`
表示了CGDB绘制的指向当前浏览代码行的箭头

`LineHighlight`
表示了当用户设置`arrowstyle`选项为`highlight`时，高亮行使用的高亮组

`Breakpoint`
表示了CGDB显示被设置了断点的行所使用的高亮组

`DisabledBreakpoint`
表示了CGDB显示断点被禁用的行所使用的高亮组

`SelectedLineNr`
表示了CGDB显示当前被选择的行。也就是光标所在的行。

`Logo`
这个高亮组是CGDB在没有源代码被自动检测到的时候所显示的logo的高亮组。

上一节：不同的高亮组，下一节：不同的颜色，目录：目录

5.2 不同的属性

CGDB支持curses提供的部分属性。它会将这些属性应用至输出窗口，但是这取决于您使用的终端是否支持这些特性。

下面列出了CGDB目前支持的一系列属性。

`normal`
`NONE`
这将会让文本保留不同样式。使用curses中的A_NORMAL属性。

`bold`
这将会让文本加粗显示。使用curses中的A_BOLD选项。

`underline`
这将会让文本带下划线显示。使用curses中的A_UNDERLINE选线。

`reverse`
`inverse`
这会将前景色与背景色调换。使用curses中的A_REVERSE选项。

`standout`
这是最适合的终端高亮模式。使用curses中的A_STANDOUT选项。

`blink`
这会使文本闪烁。使用curses中的A_BLINK选项。

`dim`
这会使文本亮度减少1/2。使用curses中的A_DIM选项。

上一节：不同的属性，下一章：CGDB键盘用户接口，目录：目录

5.3 不同的颜色

CGDB支持一些颜色，取决于您的终端支持多少种颜色。下表是一个CGDB所提供的颜色的表格。标题为NR-16的列表示终端至少支持16种颜色。标题为NR-8的列表示终端至少支持8种颜色。每种颜色对应的整数数值表示了被传入curse函数`init_pair()`的数值，该函数用来使curse创建一种新的颜色。

`COLOR NAME`	`NR-16`	`NR-8`	`NR-8 bold attribute`
Black	0	0	No
DarkBlue	1	4	No
DarkGreen	2	2	No
DarkCyan	3	6	No
DarkRed	4	1	No
DarkMagenta	5	5	No
Brown, DarkYellow	6	3	No
LightGray, LightGrey, Gray, Grey	7	7	No
DarkGray, DarkGrey	8	0	Yes
Blue, LightBlue	9	4	Yes
Green, LightGreen	10	2	Yes
Cyan, LightCyan	11	6	Yes
Red, LightRed	12	1	Yes
Magenta, LightMagenta	13	5	Yes
Yellow, LightYellow	14	3	Yes
White	15	7	Yes

上一节：不同的颜色，下一节：KUI超时选项，目录：目录

6 CGDB键盘用户接口

CGDB通过键盘用户接口从用户那里获取输入。我们通常称键盘用户接口为KUI。CGDB仅需要向KUI获取KUI提供的下一个用户输入的指令。

除了读取用户输入以及提供这些输入给CGDB以外，KUI还有两个主要的责任：它需要检测用户输入自定义的键盘映射与用户按下的特殊键。

用户定义的映射，或是简单映射，是用来改变输入的按键的含义。一些用户可能会称将这种功能称之为 宏 。例如：`map a b`。当用户输入了\字符，则KUI将会检测到并且替换为\然后将\返回给CGDB。

当用户输入了键盘上的特殊字符时，一个键码会被发往CGDB。例如HOME、DEL、\等等。当这样的键被按下时，操作系统将会发送几个字符给应用程序，而不是像普通的按键一样仅发送一个字符。这些连结的字符被称之为一个按键序列。KUI则负责将这些按键序列进行组合，并且向CGDB报告：有一个特别的按键被用户按下。ESC键是比较特殊的，因为大多数的键码都以它为开始。它通常给出了所有的按键序列的通常的头部。KUI使用了terminfo数据库去判断按键序列是由什么键码产生的。有少部分常用的按键序列被硬编码进CGDB中。

KUI主要的挑战是如何判断何时一个映射或者按键序列被输入完成。KUI有时需要读入不止一个字符去确定映射或者按键序列被输入完成。例如，用户设置了两个映射， `map abc def` 与 `map abd def`，KUI需要在它能判断用户是否要输入一个映射之前缓存\与\两个字符。当下一个键被按下时，如果用户输入\或是\则KUI收到一个映射，然后将`d e f`返回给CGDB。否则，没有映射被接收到，KUI将会把`a b`返回给CGDB。

选项 *timeout* ， *ttimeout* ， *timeoutlen* 以及 *ttimeoutlen* 可以被用来告诉KUI是否需要在映射或是按键序列的中间保持超时，以及如果需要的话，需要保持多久。

- 配置KUI的超时选项：KUI超时选项
- 理解键盘映射：使用键盘映射
- 理解键码：理解键码

上一节：CGDB键盘用户接口，下一节：使用映射，目录：目录

6.1 KUI超时选项

KUI可以被配置为在输入部分的映射或键序列时，经过一定时间间隔后超时。

当KUI匹配了一部分的映射或键序列时就可以超时。这意味着它将在前一个按键后等待一段时间，并且在超时后接收这个独立的按键。这是很显而易见的，因为用户在输入映射时必须独立的输入每个字符。对于部分的序列则并不这么明显，因为用户只敲击了一个按键，但是多个字符就会被发送至CGDB。下表描述了用户可以如何配置KUI对键码和映射的超时选项，通过 *timeout* 与 *ttimeout* 选项。

上一节：KUI超时选项，下一节：键码，目录：目录

6.2 使用映射

CGDB 完整支持键盘映射，允许用户改变键盘输入的含义。比如，你可以这样的射：`map <F2> ip<Space>argc<CR>`。

在CGDB模式下，如果按下 F2 键，就会被替换成它所映射的值。 CGDB 先会收到 i 键，进入插入模式，接着 CGDB 会收到 p argc 以及一个紧跟着的回车键。

CGDB 目前支持两个映射表。用 *map* 命令添加的映射在 CGDB 模式下会给 CGDB 使用。要删除由 *map* 命令添加的映射，可以使用 *unmap* 命令。如果想在 GDB 模式下进行映射，可以使用 *imap* 命令，而 *iunmap* 命令则可以删除 *imap* 所添加的映射。例如：

```
map ab foo
unmap ab

imap ab foo
iunmap ab
```

上一节：使用映射，下一节：对被调试的程序进行I/O操作，目录：目录

6.3 理解键码

对于不大熟悉 vim 映射的人来说前面的例子需要一点解释。映射包括一个键和一个值，它们以一个空格作为分隔符分开。键和值的表示形式均不能直接含有空格，否则这些空格会被认为是分隔符。如果需要在键或者值之内包含一个空格，可以使用键码记号 <Space> 。下面是以键码记号形式表示的键码表，这些键码记号可以在所有映射命令中使用。

记号	表示含义
<Esc>	ESC键
<Up>	光标向上移动
<Down>	光标向下移动
<Left>	光标向左移动
<Right>	光标向右移动
<Home>	HOME 键
<End>	END 键
<PageUp>	向上翻页
<PageDown>	向下翻页
<Del>	DELETE 键
<Insert>	INSERT 键
<Nul>	空 (NULL)
<Bs>	BACKSPACE 回退键
<Tab>	TAB 键
<NL>	换行
<FF>	换页
<CR>	回车
<Space>	空格
<Lt>	小于
<Bslash>	反斜杆 \
<Bar>	竖杠
<F1>-<F12>	功能键 F1 .. F12
<C->	Ctrl 控制键
<S->	Shift 键

上一节：理解键码，下一节：在CGDB中允许终端控制流，目录：目录

7 对被调试的程序进行I/O操作

如果被调试的程序需要读取终端用户输入，我们推荐用户在终端中启动被调试程序，然后在另一个终端使用CGDB去attach被调试程序，这是与被调试程序进行I/O交互最简单的方法。

然而如果用户希望能直接在CGDB内对被调试程序进行输入，CGDB也提供了这样的机制。下述的方法在Windows下编译的GDB上是不可行的。在Cygwin中运行的GDB上或许可以运行。

这个方法和进入/退出GDB模式的方法相似。在缺省的情况下，tty窗口是不可见的。这是因为用户仅在想要与被调试程序进行I/O操作时才会需要用到这个窗口。在命令模式下按 *T* 键可以显示这个窗口。在按下 *T* 键之后，你会发现在代码窗口和gdb窗口中间出现了一个新的窗口。它叫做tty窗口。同时您也会看见一个新的状态栏，叫作tty状态栏。在您键入 *T* 之后，tty状态栏上会显示出一个 '' 键。这是因为当键入 T 键，这个窗口被打开时，*CGDB*会自动将用户置入*TTY*模式。想要退出这个模式可以键入*CGDB*模式切换键（默认为*ESC*键）。这将会让您进入命令模式。可以通过在命令模式下键入 T* 键打开和隐藏tty窗口。

tty窗口被打开后，用户可以键入 *I* 命令进入TTY模式。用户可以在TTY模式下键入cgdb模式切换键退出到命令模式。

当tty窗口打开时，所有从程序输出的数据将会被显示在那里。任何在tty窗口被输入的数据将会被传输到被调试的程序中。这些数据将不会被送到GDB。当tty窗口关闭时，所有的输出将会通过GDB窗口显示出来，同时也会在tty窗口被输出（这样在tty窗口后来被打开时就可以看到之前的程序输出）。

如果用户希望为被调试程序获取一个新的tty窗口，可以键入Ctrl - T。这将会清空被调试程序的输入缓冲区中的所有数据。这个功能也许会在您重新运行被调试程序的时候有帮助。